Pwrake : A Parallel and Distributed Flexible Workflow Management Tool for Wide-area Data Intensive Computing

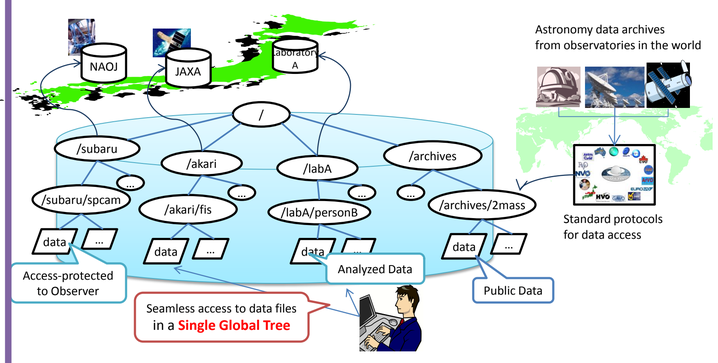
Masahiro Tanaka and Osamu Tatebe (University of Tsukuba)

ABSTRACT

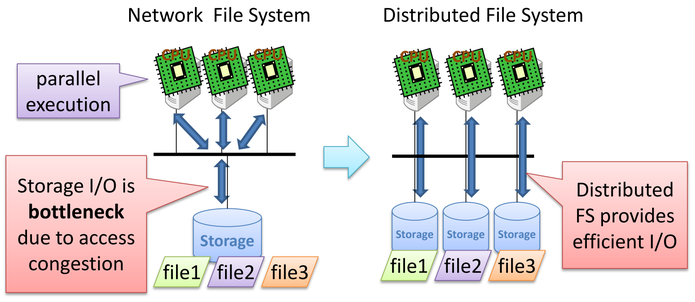
This poster proposes ***Pwrake***, a parallel and distributed flexible workflow management tool based on *Rake*, a domain specific language for building applications in the Ruby programming language. *Rake* is a similar tool to *make* and *ant*. It uses a *Rakefile* that is equivalent to a *Makefile* in *make,* but written in Ruby. Due to a flexible and extensible language feature, *Rake* would be a powerful workflow management language. The ***Pwrake*** extends *Rake* to manage distributed and parallel workflow executions that include remote job submission and management of parallel executions. This paper discusses the design and implementation of the ***Pwrake***, and demonstrates its power of language and extensibility of the system using a practical e-Science data-intensive workflow in astronomical data analysis on the **Gfarm** file system as a case study. Extending a scheduling algorithm to be aware of file locations, 20% of speed up is observed using 8 nodes (32 cores) in a PC cluster. Using two PC clusters located in different institutions, the file location aware scheduling shows scalable speedup. The extensible ***Pwrake*** is a promising workflow management tool even for wide-area data analysis.

BACKGROUND AND MOTIVATION

Data Intensive Computing in e-Science

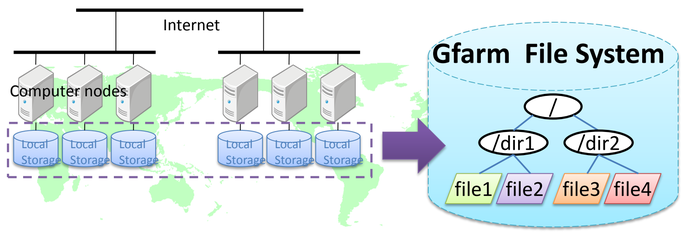


Data Intensive Computing requires Distributed File System



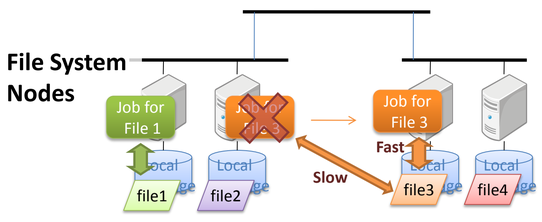
Gfarm: a Wide-area Distributed File System

- http://datafarm.apgrid.org/
- Global namespace to federate storage of compute nodes
- Designed for data intensive computing in wide area



Key issue for Scalable I/O performance: File Affinity Task Scheduling

- Exploit local I/O for scalable I/O performance
- Move and execute program instead of moving large-scale data
- So far there is no workflow tool with file affinity scheduling.



Requirement for Workflow Tool

Extensibility :
Able to choose scheduling scheme, especially affinity-aware scheduling.

Programmable :
Easy to define complicated workflows and parameter sweep.

Rule-based :
Same definition for different set of data. (DAG-based workflow is not re-usable.)

Dynamic Task Definition:
Define tasks based on the result of former tasks.

Performance :
Scalability in parallel execution.

Pwrake = *Rake* + Parallel Workflow extension

What is *Rake* ?

- ✓ A build tool similar to ***make***
- ✓ Written in Ruby language
- ✓ Part of Ruby 1.9.x

Why *Rake* ?

- ✓ Widely used as a build tool
- ✓ Easy to write complicated workflows using Ruby language features such as parameter sweep
- ✓ Easy to extend behavior by inheriting Task class
- ✓ Easy to define task dynamically

Rake syntax = Ruby syntax

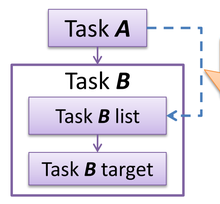
Ruby method defined in *Rake*

Key-value argument to `file` method
task_name => prerequisites

```
file "prog" => ["a.o", "b.o"] do
  sh "cc -o prog a.o b.o"
end
```

Ruby code block enclosed by `do … end` or `{…}` executed as a task action.

Dynamic Task Definition



This workflow can be defined as:

```
TASK_B_LIST = Array.new

task "A" do
  TASK_B_LIST << ...
end

task "B" => "A" do
  a = TASK_B_LIST.map do |b|
    task b do
      ...
    end
  end
  task("B-target" => a).invoke
end
```

Pwrake feature : Concurrent Workflow Execution

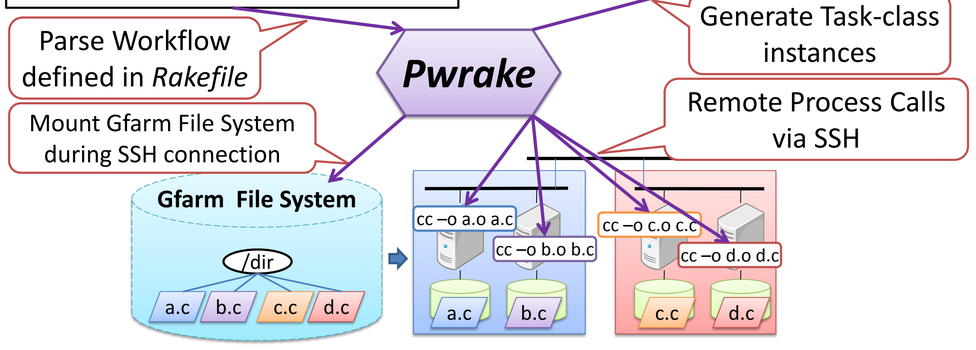
```
INFILES = FileList["?.c"]

OUTFILES = INFILES.map do |i|
  o = i.sub(/.c$/, ".o")
  file( o => i ) do |t|
    t.rsh "cc -o #{o} #{i}"
  end.name
end

pw_multitask( "target" => OUTFILES )
do
  sh "cc -o x #{OUTFILES.join(' ')}"
end
```

```
INFILES = ["a.c","b.c","c.c","d.c"]
```

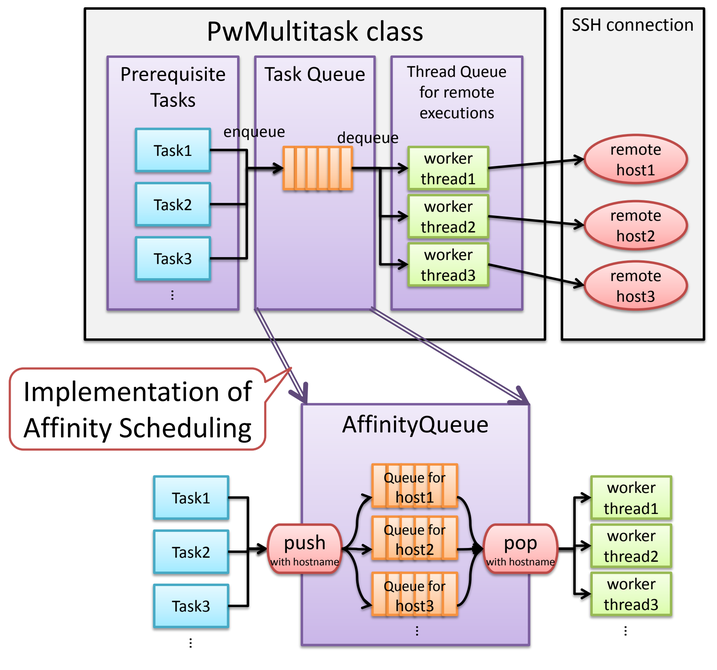
```
i = "a.c"    o = "a.o"
```

```
Task class instance :
  @name :          "a.o"
  @prerequisites : "a.c"
  @action : proc{|t| t.rsh "cc -o a.o a.c"}
```

```
PwMultiTask class instance :
  @name :          "x"
  @prerequisites : ["a.o","b.o",..]
  @action : proc{|t| sh "cc -o x a.o b.o .."}
```

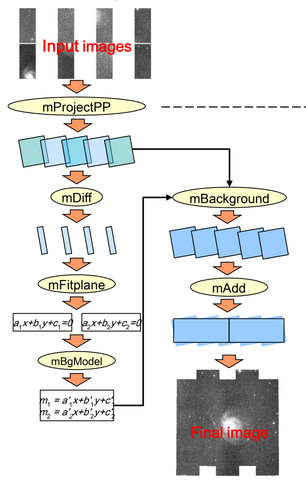


Pwrake Implementation



CASE STUDY : Astronomy Workflow

Montage workflow



mProjectPP task definition for *Pwrake*

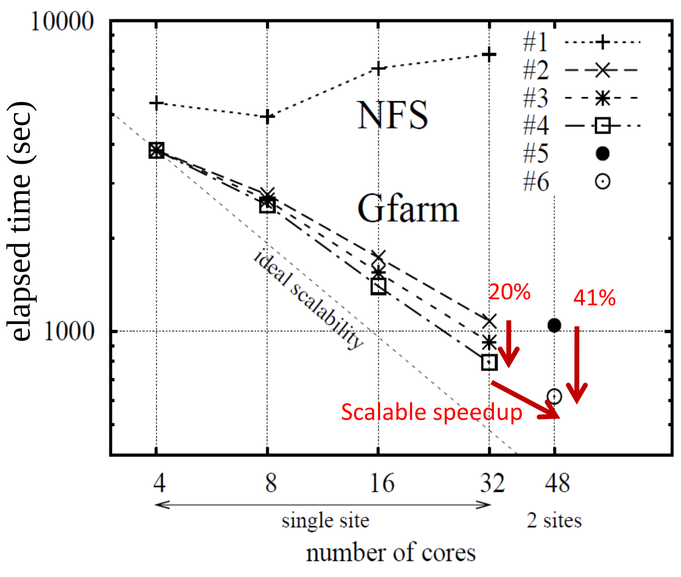
```
SRCFITS = FileList[ "#{INPUT_DIR}/*.fits" ]

file( "pimages.tbl" ) do
  OUTFITS = SRCFITS.map do |i|
    o = i.sub(/^(.*?)([^¥/]+).fits/,'p/¥2.p.fits')
     file( o => [i, HDR] ) do |t|
       t.rsh "mProjectPP #{i} #{o} #{HDR}"
     end
    o
  end
  pw_multitask( "Proj" => OUTFITS ).invoke
  sh "mImgtbl p pimages.tbl"
end
```

Performance Evaluation

- Workflow
  - Montage : a tool to combine astronomical images
  - http://montage.ipac.caltech.edu/
- Input data:
  - 2MASS All sky survey
  - 1,580 files (3.3 GB)
- Platform :

site	core	nodes	memory
Univ of Tsukuba	quad	8	4GB
AIST	dual	8	2GB

Result of Performance Evaluation



One site :

- Site: Univ. of Tsukuba

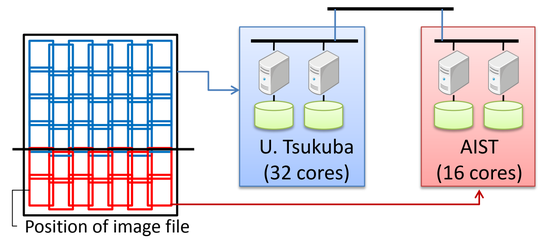
NFS (plot #1):
- Elapsed time increases even as the number of core increases.

Gfarm (plot #2-6):
- **#2** : Without Affinity scheduling
- **#3** : With **Affinity scheduling**
- **#4** : Same as #3 except **input data are distributed** across compute nodes
- All the cases show scalable speedup.
- Performance (32 cores) :
  - **#2→#3** : **14 % speedup**
  - **#2→#4** : **20 % speedup**

Two sites :

Gfarm (#5 and #6) with 48 cores
- Site: Univ. of Tsukuba and AIST
- Scheduling: Affinity scheduling (same as #3,4)
- Arrangement of input data :
  - **#5**: Each cluster has one file replica for each input file.
  - **#6**: See figure below.
- Performance :
  - **#5→#6** : **41% speedup**
- **Scalable speedup** is observed in comparison to one-site

Arrangement for **#6**: Input files are assigned to sites by celestial coordinate. It reduces file accesses between sites.



RELATED WORKS

GXP make

- A workflow management tool which exploits the GNU make and uses GXP, a parallel shell tool written in Python, as the underlying distributed execution engine.
- Define workflows in Makefile.
- It has implicit and explicit rules to execute, variable values, and shell scripts.
- It is possible to reduce the length of a workflow description dramatically compared to the DAG input file, and to generate a general workflow for applications. This research is inspired by the GXP make.

Swift

- A scientific workflow system designed for loosely coupled computations.
- Define workflows in a statically typed language called SwifScript.
- Swift dispatches a workflow to another scheduler, such as Karajan, while it is not intended for users to extend the scheduler. Such batch job submission needs granularity of jobs for efficient execution.

CONCLUSION

- ***Pwrake***, a parallel and distributed flexible workflow management tool, is proposed.
- ***Pwrake*** is extensible, and has flexible and powerful workflow language to describe scientific workflow.
- We demonstrate a practical e-Science data-intensive workflow in astronomical data analysis on **Gfarm** file system in wide area environment.
- Extending a scheduling algorithm to be aware of file locations, 20% of speed up was observed using 8 nodes (32 cores) in a PC cluster.
- Scalable speedup is observed in the measurement using two PC clusters located at different sites, if each file is grouped by coordinate and placed at an appropriate site based on the group.